(12) UK Patent Application (19) GB (11) 2 372 679 (13) A

(43) Date of A Publication 28.08.2002

(21) Application No 0104788.5

(22) Date of Filing 27.02.2001

(71) Applicant(s)
AT&T Laboratories-Cambridge Limited
(Incorporated in the United Kingdom)
Old Addenbrookes Site, 24A Trumpington Street,
CAMBRIDGE, CB2 1QA, United Kingdom

(72) Inventor(s)
Steven Leslie Pope
Derek Edward Roberts
David Clarke

(74) Agent and/or Address for Service
Marks & Clerk
4220 Nash Court, Oxford Business Park South,
OXFORD, OX4 2RU, United Kingdom

(51) INT CL[7]
H04L 12/46

(52) UK CL (Edition T)
H4P PF

(56) Documents Cited
GB 2352146 A
WO 1997/008838 A2
US 5651002 A
EP 0594199 A1
US 6108345 A

(58) Field of Search
UK CL (Edition S) **H4P PF PPA PPEC**
INT CL[7] **H04L 12/46 12/66 29/06**
ONLINE DATABASES: WPI, EPODOC, JAPIO.

(54) Abstract Title
Network Bridge and Network

(57) A network bridge (10, fig. 2) is provided for interfacing between a first network having a first protocol, and a second network having a second protocol. The network bridge has a buffer (29, fig. 2) which receives data packets and a buffer (30, fig. 2) which supplies headers corresponding to the final destination address in memory to which an application for processing the payload data of the packet requires delivery. The packets are assembled at (31, fig. 2) so that the payload data are delivered to the final destination address without the need for a protocol stack in a computer of the second network. When passing data from the second network to the first, data packets are simply assembled from data payloads in accordance with the second protocol and headers already in accordance with the first protocol, requiring no "look-up" action.

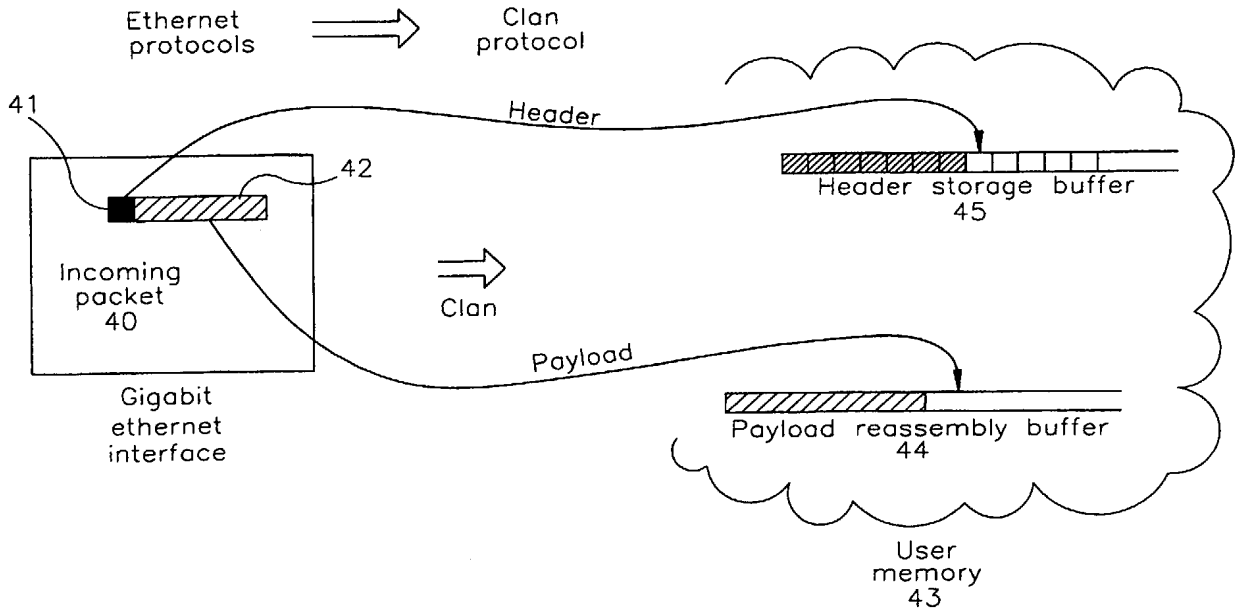


FIG 3

GB 2 372 679 A

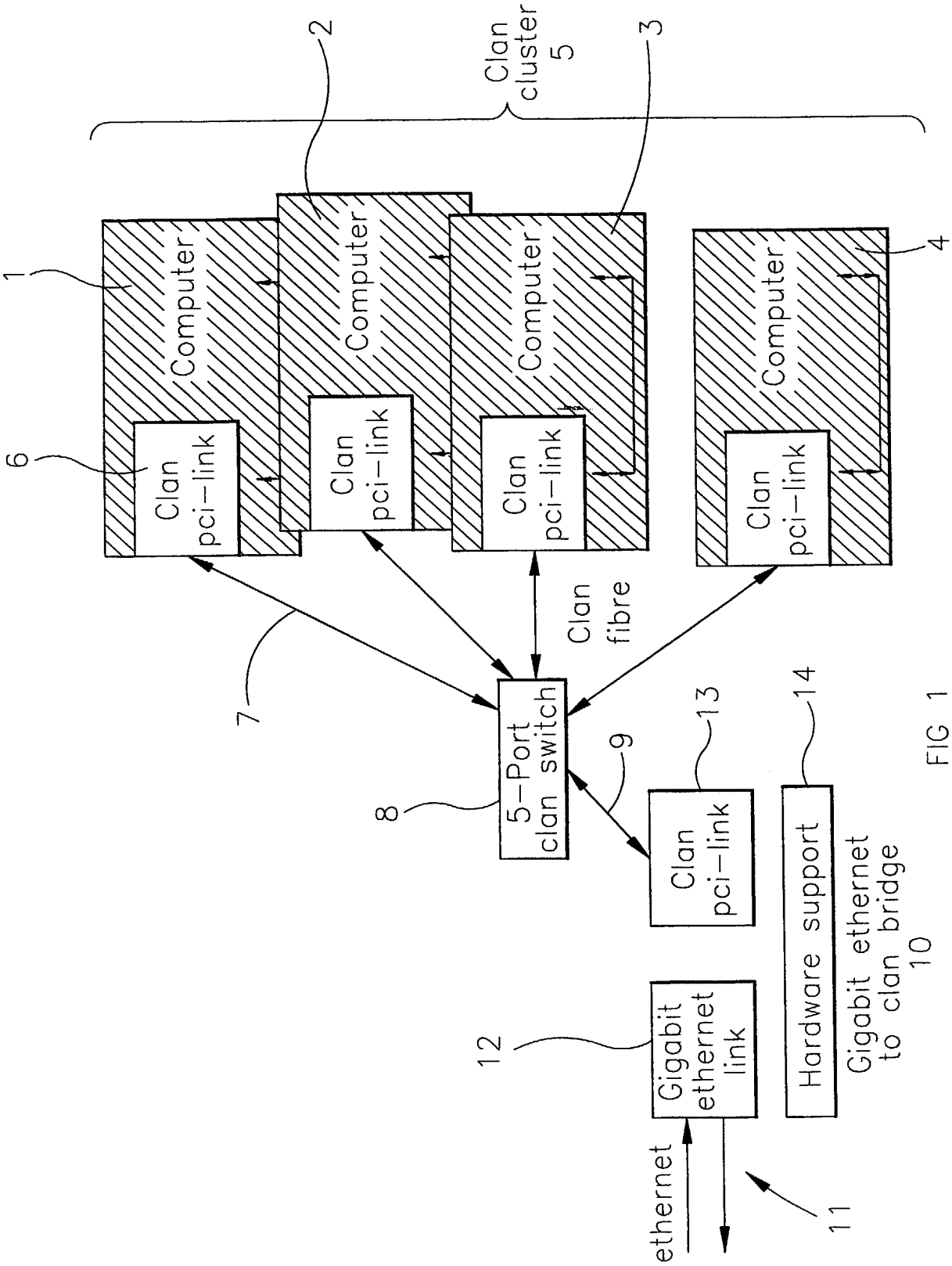

Clan cluster 5
2
3
1
Computer
Computer
Computer
4
Computer
6
Clan pci–link
Clan pci–link
Clan pci–link
Clan pci–link
Clan fibre
7
5–Port clan switch
8
9
13
Clan pci–link
14
Hardware support
Gigabit ethernet to clan bridge 10
12
Gigabit ethernet link
ethernet
11

FIG 1

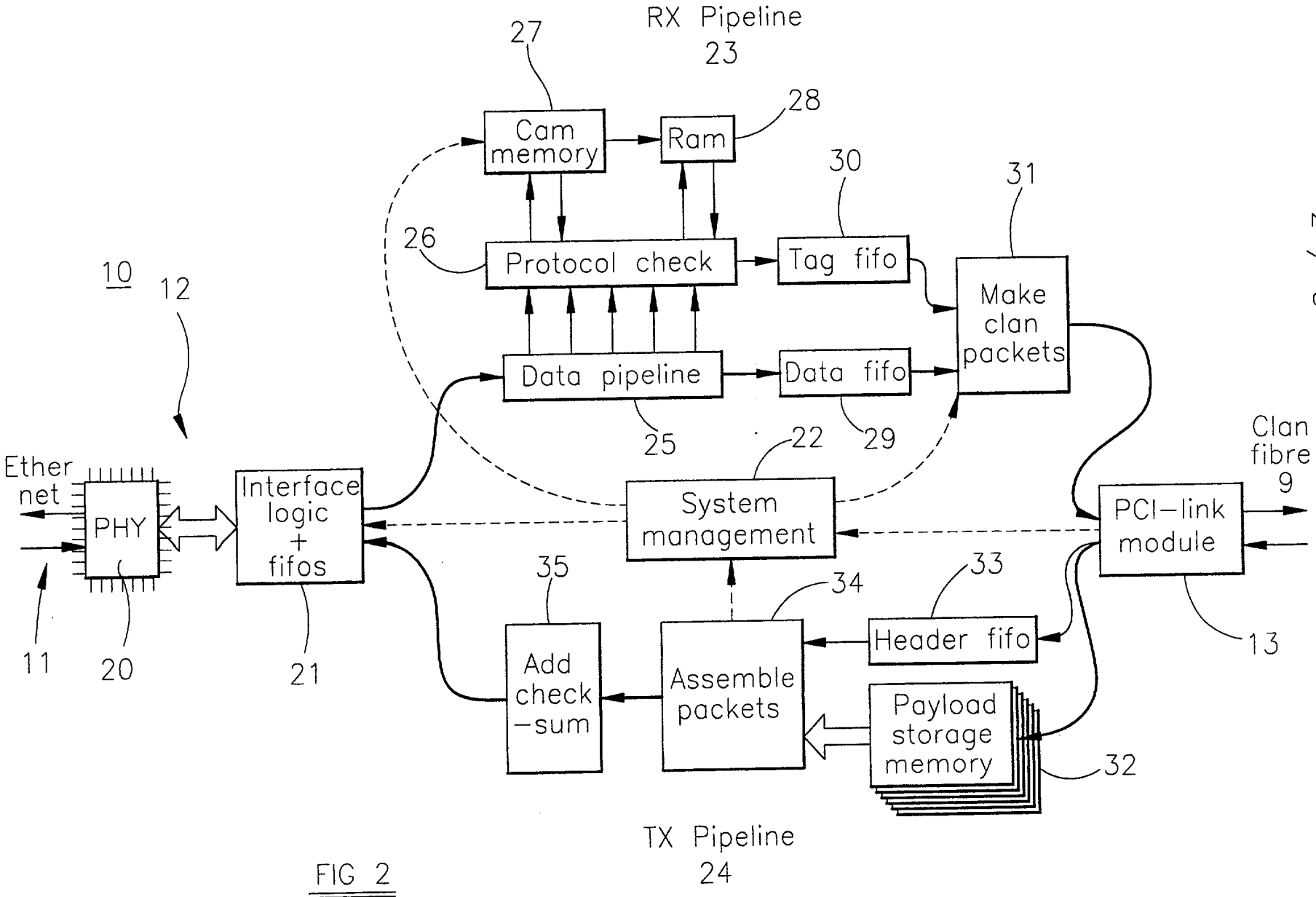

RX Pipeline
23
27
28
Cam memory
Ram
30
31
26
Protocol check
Tag fifo
Make clan packets
10
12
Data pipeline
Data fifo
25
29
22
Clan fibre 9
Ether net
PHY
Interface logic + fifos
System management
PCI–link module
35
34
33
Header fifo
11
20
21
Add check –sum
Assemble packets
Payload storage memory
13
32
TX Pipeline
24

FIG 2

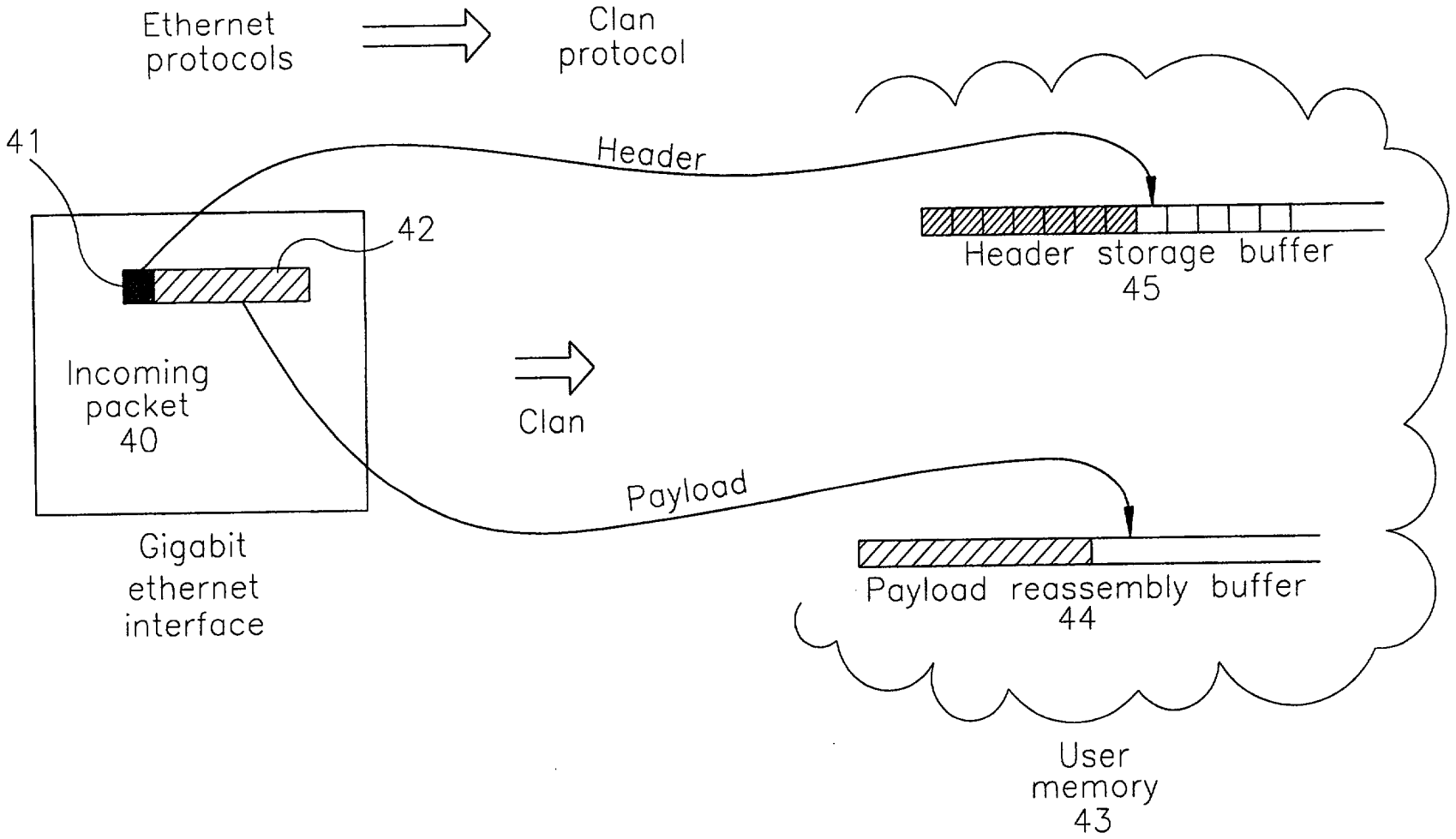

Ethernet protocols
Clan protocol
41
Header
42
Header storage buffer
45
Incoming packet
40
Clan
Payload
Gigabit ethernet interface
Payload reassembly buffer
44
User memory
43

FIG 3

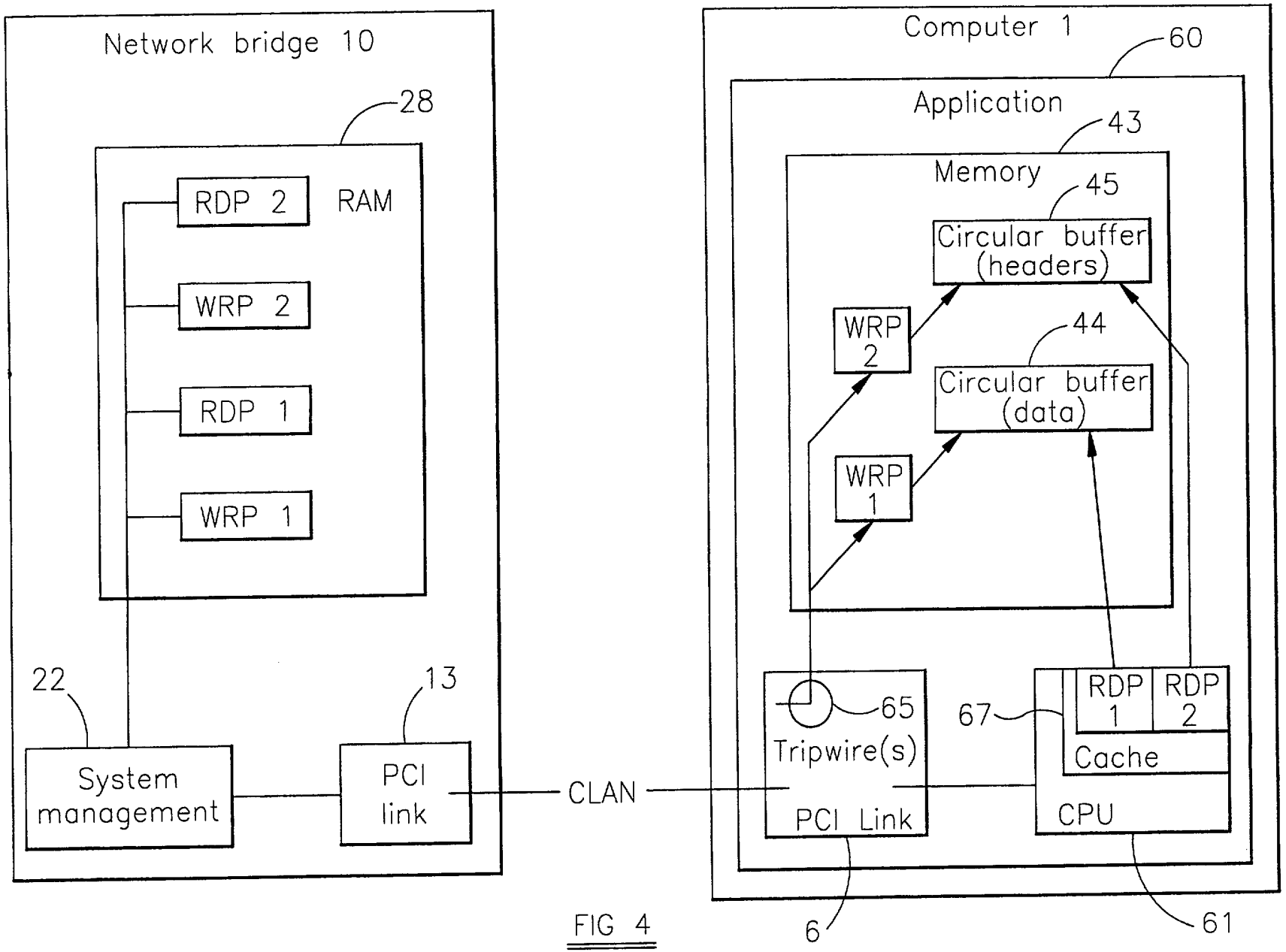
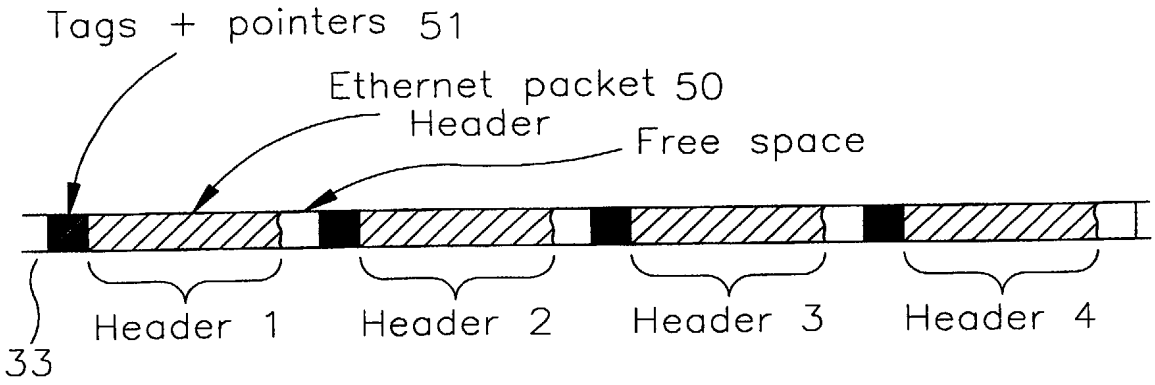

Network bridge 10
28
RDP 2
RAM
WRP 2
RDP 1
WRP 1
22
13
System management
PCI link
CLAN
Computer 1
60
Application
43
Memory
45
Circular buffer (headers)
WRP 2
44
Circular buffer (data)
WRP 1
65
Tripwire(s)
PCI Link
67
RDP 1
RDP 2
Cache
CPU
6
61

FIG 4


Tags + pointers 51
Ethernet packet 50
Header
Free space
Header 1
Header 2
Header 3
Header 4
33

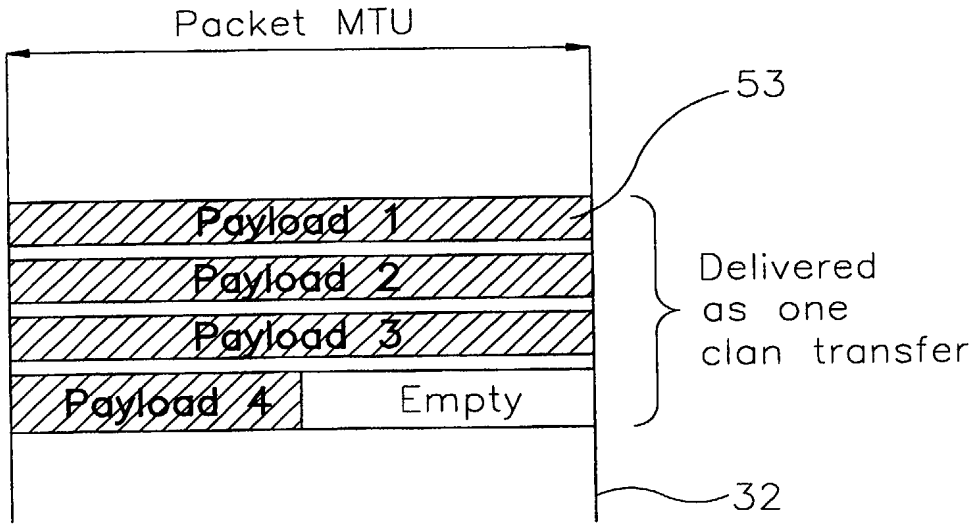

Packet MTU
53
Payload 1
Payload 2
Payload 3
Payload 4
Empty
Delivered as one clan transfer
32

FIG 5

Network Bridge and Network

The present invention relates to a network bridge for interfacing between a first network having a first protocol and a second network having a second protocol different from the first protocol. The present invention also relates to a network, such as a collapsed local area network (CLAN), including such a network bridge.

A CLAN is disclosed in "Tripwire: A Synchronisation Primitive for Virtual Memory Mapped Communication", 4th Intl Cnf on Algorithms and Architectures for Parallel Processing, 2000, (ICA3PP) ftp://ftp.uk.research.att.com:/pub/docs/att/tr.2000.7.pdf and connects computers together using a high speed optical fibre and special switch hardware. When computers of a group or "cluster" are connected in this way, they are able to exchange information very quickly and with low latency. This allows certain tasks to be performed with super-computer speed on standard "commodity" hardware.

Many possible applications require that a CLAN cluster be connected to other computers using other standard networking technology. Known protocols such as TCP and UDP are natural choices for such networking technology because they are protocols which are used by the Internet and by the majority of existing private data networks. Both of these protocols run over another protocol known as Internet Protocol (IP). IP may be run on many types of hardware but Ethernet is the most commonly used and generally the most affordable. Ethernet is disclosed by R.Metcalfe and D.R.Boggs in Communications of the ACM, "Ethernet: Distributed packet switching for local computer networks", Vol.19 No.5 July 1976, the contents of which are incorporated herein by reference. Ethernet originally ran at a data rate of a few megabits per second on special coaxial cable for sending frames or packets of data between computers. Ethernet is in widespread current use with the same packet format and access rules but the wiring has improved and the interfaces are much faster. Most current installations support operation at 10 megabits per second and 100 megabits per second and use unshielded twisted pair "UTP" cable. The more recent Gigabit Ethernet uses the same packet sizes and access rules but at a bit rate of 1000 megabits (1gigabit) per second. The original Ethernet was used as a shared bus but it is nowadays usual to use an

Ethernet switch and star wiring to maximise the available bandwidth to and from the individual computers making up the network.

In a known type of Ethernet network, traffic to and from the network is handled inside each computer of the network by a software module known as a "protocol stack". When a data item is to be sent, it is transferred to the protocol stack, which then slices it into appropriately sized "pieces". Each piece is "wrapped up" as an Ethernet packet by adding suitable protocol headers. If the data item is large and fills many packets, the protocol stack can do this efficiently by packetising the whole data item in one go.

In contrast, incoming traffic is a mix of packets from different computers for re-assembly into different data items. The protocol stack has to check each packet header carefully and copy the packet data payload into the correct place in a suitable buffer.

When a computer is using a 10 megabits per second or 100 megabits per second Ethernet, the network itself presents a bottle neck which limits the amount of central processing unit (CPU) time required by the protocol stack. However, with the Gigabit Ethernet, the network is no longer the limiting factor and the maximum achievable rate of transfer of data which is actually achieved is usually about 300 megabits per second with most of the CPU time being consumed by running the protocol stack.

Many things conspire to waste CPU time when dealing with a network but the main contenders are:

Copying packet data from one buffer to another;

Process switching; and

Interrupt handling.

Copying of packet data takes place at the speed of main memory, which is many times slower than the cache memory used by the CPU for calculations.

Process switching occurs when a process (also known as an application) running on a computer is ended or suspended and replaced by another process or application. In order to be able to restore operation of a process or application, the state of the CPU has to be stored when a process is suspended and restoring a previously suspended process requires restoring the CPU state. This may also require a pause while the CPU waits for the cache memory to be refilled.

Interrupts are triggered by events such as packet arrival and force switches into and out of an interrupt handler code which takes over control of the CPU when an interrupt has been requested.

A CLAN has some similarities to a Gigabit Ethernet network. In particular, both types of network have gigabit links, the computers of the cluster are star-wired to a switch and data are transferred in the form of packets. However, CLAN packets are substantially different from Ethernet packets. In particular, CLAN packets are effectively PCI bus transfers which take place over the CLAN hardware from one computer into the memory of another computer. Thus, CLAN packets do not have the large and complex headers which typify Ethernet packets and which have to be unravelled in order to perform the packet transfer.

The simplest way to connect a CLAN cluster to a Gigabit Ethernet would be to dispose an Ethernet card in one of the computers of the cluster and to use it as a gateway. However, such an arrangement would suffer from all of the performance problems mentioned hereinbefore.

Some of these limitations could be overcome by arranging for an interface to have its own CLAN link to a switch so that Ethernet packets can be routed via the CLAN to separate protocol stacks in all of the computers of CLAN cluster. Such an arrangement would still require the same amount of packet processing but the load could be spread more evenly among the computers of the cluster rather than occupying most or all of the CPU time of one computer. In particular, each computer would have some CPU time

left to do useful work. However, such an arrangement has little advantage over disposing an Ethernet card in each computer and connecting them to an Ethernet switch.

Ethernet interface cards use sections of computer memory called "buffers" to queue up incoming and outgoing packets. The buffers are filled and emptied by the interface hardware using direct memory access (DMA). When packets have been transmitted so that the transmitter buffers have become free or empty or when new packets have arrived so that the receive buffers have been filled up, the network interface card requests an interrupt.

When the CPU of the computer receives an interrupt request, it is forced to perform the immediate execution of the special interrupt handler code. It is usual for the interrupt handler code to be non-trivial so that time has to be spent in saving and restoring CPU states.

Interrupts are necessary because process scheduling is managed entirely in software. Computers do not usually provide a hardware mechanism for making a process runnable when a logical condition is true so this is generally handled by means of the interrupt mechanism.

There are some known strategies for reducing the impact of interrupts from a network interface. For example, some network interface cards allow a limit to be placed on the rate at which interrupts may be requested. This provides some improvement but at the cost of a small increase in latency.

An alternative to using the interrupt mechanism is to provide an arrangement which periodically polls the network interface to determine whether there has been a change in a buffer state. However, such polling can be very wasteful if the workload fluctuates unpredictably.

Some network interfaces use special hardware or built-in firmware to perform "packet coalescing". This technique joins together the payloads of suitable packets and applies

a new header to make the resulting packet legal. In some circumstances, the reduced packet count can improve performance. However, this technique is not very helpful where the incoming packets are from many different sources and contain fragments of many different data items.

Where a computer is running several processes and it is required to pass data from one process to another, this can be achieved without copying the data from one part of memory to another part. Instead, the "ownership" of the data can be transferred such that the ownership of the physical memory occupied by the data is passed from one process to another. The "physical" copying of data from one place to another in memory is much more time consuming. However, there are two occasions when this is required by known networks, namely when the network interface places a packet in a packet buffer and when the protocol stack copies payload data into a re-assembly buffer. Placing a packet into a buffer corresponds to delivery of the packet and so cannot be avoided.

Operating systems allow many separate "user processes" to be run on a single computer by sharing the CPU time between the processers. A privileged process called the "kernel" controls the time sharing so as to allow each user process in turn to run for a short time. Individual user processes are written as if they had sole use of the computer. The kernel works hard to make sure that breaks in execution have no effect on the logical flow of each user process.

A process switch can occur at any time, for example even in the middle of evaluating a group of interdependent numbers, so that it is unsafe to allow a user process to alter a shared resource such as a packet queue. For this reason, it is conventional to run a single network protocol stack in the kernel, where it can be sure of uninterrupted operation.

Such an arrangement is inefficient because each data item sent to or received from the protocol stack causes a process switch into and out of the kernel. As mentioned hereinbefore, time is wasted saving and restoring CPU states and a process switch often

provokes longer delays while the cache memory is being refilled. For example, a typical process switch may take of the order of 5 microseconds and a further 10 microseconds may be required if a cache refill is necessary.

According to a first aspect of the invention, there is provided a network bridge for interfacing between a first network having a first protocol and a second network having a second protocol different from the first protocol, comprising: first means for receiving data packets from the first network each data packet having a header and a data payload in accordance with the first protocol; and second means for adding to each of at least some of the received packets delivery data in accordance with the second protocol indicating the final destination address to which an application for processing the payload data of the packet requires delivery.

The first destination address may be the address in a randomly addressable device, such as a memory.

The first network may be an Ethernet, such as a Gigabit Ethernet. As an alternative, the first network may be an ATM network.

The second network may be memory mapped network such as a collapsed local area network.

The second means may be arranged to remove at least part of the header from each of the at least some received packets. The at least part of the header may comprise address data. The at least part of the header may comprise all of the header.

The second means may be arranged to process at least part of the header of each of the at least some received packets so as to generate the delivery data. The second means may contain a mapping from the at least part of the header of each of the at least some received packets to the delivery data, which mapping is programmable by at least one application in the second network. The second means may comprise a content

addressable memory arranged to address a random access memory. The at least part of the header may include a sequence number of the payload data.

The second means may be arranged, for the or each application, to direct the payload data of received packets to the correct position, relative to the positions of other payload data for the same application, in a payload reassembly buffer of a computer on which the application runs, which re-assembly buffer is the final destination for the payload data. The second means may be arranged to direct the headers of the received packets to a header storage buffer of a computer on which the application runs for storage in order of arrival of the received packets.

The network bridge may comprise: third means for receiving from the second network a plurality of headers in accordance with the first protocol; fourth means for receiving, independently of the headers, payload data from the second network; and fifth means for assembling the headers and the payload data into packets for transmission to the first network.

The third means may comprise a first-in/first-out buffer.

Each header received from the second network may include a pointer to a location in the fourth means for the corresponding payload data. The fifth means may be arranged to remove the pointers before transmission to the first network.

The network bridge may comprise sixth means for calculating a checksum for each packet and for adding the checksum to the header.

According to a second aspect of the invention, there is provided a combination of a network bridge according to the first aspect of the invention and a network switch for the second network having a first port connected to the network bridge.

The network bridge may be connected to the switch by an electrically conductive signal path.

According to a third aspect of the invention, there is provided a network comprising a combination according the second aspect of the invention and a plurality of computers, each of at least some which is connected to a respective further port of the switch and has a final destination memory.

Each of the at least some computers may be connected to the switch by a respective optical fibre.

At least one of the computers may be arranged to run a first application which generates at least some of the packet headers in accordance with the first protocol. The first application may be arranged to reserve space in the fourth means for the payload data corresponding to the at least some packet headers in accordance with the first protocol. The first application may be arranged to supply the payload data by direct memory access. The first application may be arranged to request another application and/or to a device which is external to a central processing unit for running the first application, such as a disk controller, to supply the payload data by direct memory access

At least one of the computers may be arranged to run a second application which requires data from the first network. The second application may be arranged to provide a circular buffer for receiving payload data from the network bridge. The application may be arranged to provide a write pointer indicating a next position in the circular buffer for receiving the payload data and a read pointer indicating a position in the circular buffer where next data to be read are located. The payload reassembly buffer may be the circular buffer and the network bridge may be arranged to update the write pointer when the circular buffer contains a contiguous block of data.

The second means may be arranged to generate the delivery data from the write pointer and a packet sequence number. The at least one computer may comprise means for informing the application when the write pointer is updated, for example means for

requesting an interrupt or means for setting a flag which is periodically polled by the application.

The network bridge may be arranged to compare the read and write pointers and to stop sending payload data to the circular buffer when the circular buffer is full.

According to a fourth aspect of the invention, there is provided a network bridge for interfacing between a first network having a first protocol and a second network having a second protocol different from the first protocol, comprising: first means for receiving from the second network a plurality of packet headers in accordance with the first protocol; second means for receiving, independently of the headers, payload data from the second network; and third means for assembling the headers and the payload data into packets for transmission to the first network.

According to a fifth aspect of the invention, there is provided a combination of a network bridge according to the fourth aspect of the invention and a network switch for the second network having a first port connected to the network bridge.

According to a sixth aspect of the invention, there is provided a network comprising a combination according to the fifth aspect of the invention and a plurality of computers, each of at least some of which is connected to a respective further port of the switch.

It is thus possible to provide a network bridge and a network with reduced latency. Within the second network, the payload data can be sent directly to the final destination, for example the position within memory at which an application for processing the data requires the packet to be delivered. Such an arrangement thus allows protocol stacks in user processes or applications to interact safely with a shared network interface and avoids the need for copying data from a protocol stack, for example in a kernel, to a re-assembly buffer within an application or process. The amount of CPU time in each computer of the second network required for processing data packets can be substantially reduced so that more CPU time is available for running the processes or applications.

The invention will be further described, by way of example, with reference to the accompanying drawings, in which:

Figure 1 is a schematic diagram of a network and a network bridge constituting an embodiment of the invention;

Figure 2 is a schematic diagram illustrating in more detail the network bridge of Figure 1;

Figure 3 is a diagram illustrating transfer of data packets from the network bridge to a user memory;

Figure 4 is a schematic diagram illustrating the use of circular buffers in an application and tripwires to control data transfer; and

Figure 5 is a diagram illustrating the storage in the network bridge of data received from applications running in the network of Figure 1.

Figure 1 illustrates a collapsed local area network (CLAN) which comprises a group of (in this case four) computers 1,2,3 and 4 known as a "CLAN cluster" 5. Each of the computers contains a CLAN PCI-link , such as 6, which provides interfacing of the computer to an optical fibre such as 7. The optical fibres 7 from the computers are connected to respective ports of a five port CLAN switch 8. This is a relatively simple example of a CLAN for the purpose of illustration; CLANs may contain any number of computers and there may be more that one CLAN switch.

A fifth port of the switch 8 is connected by an optical fibre 9 to a network bridge 10, which forms a bridge between the CLAN and a Gigabit Ethernet indicated at 11. The bridge 10 comprises a Gigabit Ethernet link 12, a CLAN PCI-link 13 and hardware support 14.

The network bridge 10 is shown in more detail in Figure 2. The Gigabit Ethernet link 12 comprises a physical layer integrated circuit 20 (such as a Marvell Alaska 88E1000) which is capable of dealing with the gigabit rates required to drive the physical medium, such as a UTP cable or optical fibre. Interface logic and first-in/first-out buffers (FIFO) 21 (such as a programmed field programmable gate array (FPGA) or a Gig-E Mac chip such as a Galileo GT-48520) provide all of the functions which are necessary to comply with the standards for the Gigabit Ethernet 11. As an alternative, the circuits 20 and 21 could be implemented by means of a single custom application specific integrated circuit (ASIC).

The PCI-link module 13 provides interfacing with the fibre 9 and is connected to a system management block 22 which controls operation of parts of the network bridge 10. The system management is controlled by incoming CLAN data packets from remote user processes or applications running on the computers 1,2,3 and 4 of the CLAN cluster 5.

Data packets received from the Ethernet 11 are supplied by the block 21 to a receive (RX) pipeline 23, which performs a type of protocol translation and supplies CLAN data packets to the module 13 for delivery to user applications running on the computers 1,2,3 and 4 of the network. Similarly, data packets received from the CLAN are supplied by the module 13 to a transmit (TX) pipeline 24 where they are assembled into Gigabit Ethernet packets and sent to the block 21 for delivery via the Gigabit Ethernet 11.

The RX pipeline 23 comprises a data pipeline 25 which receives Ethernet packets from the Ethernet link 12. The data in the data pipeline 25 are supplied to a protocol check block 26 which searches the packet headers for TCP, IP and UDP information in order to extract key address data and sequence numbers from the headers. Extracted addresses and sequence numbers are supplied to a content addressable memory (CAM) 27 whose output is connected to the address input of a random access memory (RAM) 28. Applications which are awaiting packets from the Ethernet 11 send packet address data and delivery address data via the CLAN and the module 13 to the system

management block 22. The block 22 writes each packet address in the CAM 27 and the corresponding delivery address (i.e. the final destination address or a base address from which the bridge can calculate the final destination address) in the RAM 28. When a received packet has an address which matches any of the packet addresses stored in the CAM 27, the CAM 27 supplies an index at its output which addresses the location in the RAM 28 of the corresponding delivery address. The CAM 27 and the RAM 28 thus contain a mapping from the addresses in the packet headers to the final destination addresses.

The headers and payload data are separated in the data pipeline 25 and supplied for separate delivery across the CLAN to a data first-in/first-out buffer 29. The protocol check block 26 supplies delivery address data from the RAM 28 to a tag first-in/first-out buffer 30 in the form of destination addresses for the individual packets in the buffer 29. Thus, the buffer 30 effectively contains CLAN "headers" and these are assembled together with the new data payloads from the buffer 29 in a block 31 so as to form CLAN packets, which are then sent via the module 13 to the CLAN for delivery directly to the applications running on the computers 1 to 4.

The TX pipeline 24 comprises a payload storage memory 32 which receives the Ethernet packet data payloads from the applications via the CLAN and the module 13. Ethernet headers are supplied separately to a header first-in/first-out buffer 33, which headers also contain pointers to the location in the memory 32 of the corresponding data payloads. The Ethernet packets are assembled in a block 34 and a block 35 forms a checksum of each complete Ethernet packet from the block 34 and adds the checksum to the header so as to provide a packet which conforms fully to the Ethernet protocol being used. The Ethernet packets are then sent to the Ethernet 11 by the Gigabit Ethernet link 12.

Figure 3 illustrates the operation of the network bridge for a packet received from the Ethernet and illustrates the delivery of corresponding data to a user application running on one of the computers of the CLAN cluster 5. The incoming packet 40 contains a header 41 in accordance with the appropriate Ethernet protocol and a data payload 42.

The incoming packet 40 is supplied to the data pipeline 25 and the block 26 extracts the address and sequence number information from the header 41. The extracted address and sequence number are supplied to the CAM 27, which, together with the RAM 28, has been updated by the applications running in the CLAN cluster 5 with a mapping for sending each recognised packet to the final destination address in a user memory 43 of the application which requires the data in the packet. Each application has its own buffers 44 and 45, but the buffers for some or all of the applications may be provided in a server. When a match is found in the CAM 27, the RAM 28 returns an address which comprises the actual final memory destination address for the received packet and specifies a payload re-assembly buffer 44 within the memory of the computer running the application which requires the data, together with the relative position in the buffer 44 in accordance with the sequence number extracted by the block 26 from the received packet. The data payload is supplied to the buffer 29 and the corresponding CLAN final destination address is supplied to the buffer 30 so that, when the CLAN packet is assembled in the block 31, it is transmitted via the CLAN directly to the user memory 43 and, in particular, directly to the correct position in the re-assembly buffer 44 where the application requires delivery of the payload data.

In some applications, the Ethernet header may be discarded completely and may simply be stripped from the received packet in the data pipeline 25. However, other applications may require that the Ethernet headers be supplied to the user memory 43. In such cases, the data pipeline 25 separates the headers 41 and supplies these to the buffer 29 as separate independent data payloads. The memory 27 supplies via the protocol check 26 the corresponding final destination address of the header to the buffer 30. Thus, the Ethernet header forms the data payload of an independent CLAN packet whose header is the final destination address. Typically, as shown in Figure 3, the resulting "header" packet is sent directly to a header storage buffer 45 within the user memory 43 and the headers are stored in the buffer 45 in the order in which they arrive with their packets at the network bridge 10 from the Ethernet 11. The Ethernet headers are thus available, within the application which received the data payload (or elsewhere). For example, the headers are available for checking and diagnostic purposes. In another example, the headers are used to allow the application to perform

a higher level protocol, such as TCP. The higher level protocol may even be performed by another application which may even run on another CPU or hardware device, to which the headers may be delivered from the bridge.

It is thus not necessary for each application running in the cluster 5 to have a complete protocol stack. In particular, whenever the addressing data in an Ethernet packet header find a match in the memory 27, the network bridge 10 ensures that the payload data are directed to the final destination location in the correct memory of the correct computer of the cluster 5. However, it is generally desirable to retain a modified protocol stack within each process. Thus, in the case of Ethernet packets where the protocol check 26 fails to detect a match within the memory 27, a tag is generated to direct the whole Ethernet packet to the appropriate protocol stack. In particular, the MAC or IP address can be used to determine which computer of the cluster 5 was the intended destination for the Ethernet packet.

The protocol check 27 also performs checking to indicate if there is an error in a packet header or in the checksum within the header. If such an error is found, the packet may be discarded by setting a flag bit in the tag. Such faulty packets may optionally be forwarded to a special buffer for debugging purposes.

The network bridge 10 may keep a record of sequence numbers for each data item which has been packetised and transmitted as separate packets via the Ethernet 11. When the bridge detects that a complete data item has been successfully re-assembled, it may notify the application which is awaiting the receipt of the data item for processing. This may, for example, be achieved by means of the techniques known as "tripwires" and disclosed in WO00/67131, the contents of which are incorporated herein by reference. An example of the use of such techniques is illustrated in Figure 4.

The computer 1 is illustrated as running an application 60 on the computer central processing unit (CPU) 61. The application 60 is awaiting data from the Gigabit Ethernet 11 and allocates space in the memory 43 in the form of a circular buffer 44 for the payload data and a circular buffer 45 for the Ethernet headers. Write pointers WRP1

and WRP2 and read pointers RDP1 and RDP2 define the positions in the buffers 44 and 45, respectively for writing and reading. In particular, each write pointer indicates the next position within each circular buffer which may receive data from the network bridge 10 and each read pointer points to the position within the circular buffer where the next data to be read are located. The read and write pointers are supplied to the network bridge 10 and are stored in the RAM 28. The RAM 28 thus contains data which allow the system management block 22 to determine the space available in the buffers 44 and 45 for the receipt of data.

When the network bridge 10 receives an Ethernet packet destined for the application 60, as described hereinbefore, it forms a CLAN packet whose address specifies the destination in the memory 43 for the payload data (and for the header if this is to be sent to the application). In particular, the protocol check 26 retrieves the address in the form of the write pointer WRP1 from the memory 28 and checks the sequence number so as to determine the correct position for the data within the circular buffer 44. Thus, even if Ethernet packets containing payload data for a particular data item are not received in the correct order, the network bridge 10 ensures that the payload data are transferred to the correct position within the circular buffer 44. The headers are stored in order of arrival in the circular buffer 45 and the write pointers WRP1 and WRP2 are updated in the memories 28 and 43 by the network bridge 10. However, the write pointer WRP1 for the data circular buffer 44 is only updated when the circular buffer 44 contains a contiguous block of data. Thus, the write pointer WRP1 is only updated when there are no missing packets.

The PCI link 6 of the application 60 is shown as having one or more tripwires 65 which monitor the updating of the write pointers WRP1 and WRP2. As described in WO00/67131, the tripwires 65 may be set to determine any appropriate event or condition. For example, a tripwire may be set to respond to updating of either of the write pointers WRP1 and WRP2 so as to inform the application 60, for example, that data (without any gaps) are available in the circular buffer 44 for reading. The tripwire may, for example, cause an interrupt or may set a flag which is periodically polled by the application 60 to check whether there are any fresh data available for reading.

The CPU 61 has a cache memory 67 which contains the read pointers RDP 1 and RDP 2. These pointers are updated whenever data are read from the buffers 44 and 45 and the updated read pointers are copied to the memory 28 in the network bridge 10.

The system management block 22 monitors the read and write pointers to determine how much space is available in the circular buffer 44 (and possibly in the circular buffer 45). If the network bridge 10 determines that either or both of these buffers is full, the network bridge 10 discards further Ethernet packets destined for the application 60. This mechanism thus limits the traffic on the CLAN. Other mechanisms may then be used to signal to the source of the Ethernet packets that the discarded packets have not been successfully delivered, for example so that they can be rescheduled for transmission. As an alternative to discarding further Ethernet packets, the network bridge 10 may deliver such packets to a kernel protocol stack or to another CPU in which the application may be replicated to as to perform load balancing.

It can sometimes happen that a packet is lost in transit. This results in a "gap" in the re-assembly buffer 44 where the data are incorrect. This may be detected within the bridge 10 or within the user application which is waiting for the arrival of the complete data item. The protocol stack within the application may then use higher-level protocols to request re-sending of the missing data. As an alternative where, for example, the headers are sent to another application, CPU or other hardware device (as mentioned hereinbefore), the other application or hardware device may support the higher level protocol (such as TCP) and may request resending of any missing packet.

The network bridge 10 handles the transmission of packets to the Ethernet 11 from all of the applications running on the computers of the CLAN cluster 5. Each application generates the appropriate Ethernet header and supplies this via the CLAN to the buffer 33 of the bridge 10. This FIFO buffer thus arranges for packets to be sent in order of arrival of the corresponding headers from the applications. The payload data for the Ethernet packets are sent separately (for example by means of direct memory access from a memory of the application or of another application or from an external device

such as a disk controller) by means of a CLAN transfer. The application will previously have requested the appropriate space in the memory 32 and a pointer to the memory space is included in the Ethernet header sent to the buffer 33. Thus, no copying of payload data for transmission is necessary within the application.

The block 34 assembles the Ethernet packets by attaching the relevant payload data from the memory 32 to each header in turn at the output of the buffer 33. The block 35 calculates and adds the appropriate checksum for the desired Ethernet protocol and the resulting packet is transmitted to the Ethernet 11. The system management clock 22 then informs the application that the packet has been transmitted and indicates that the memory space which contained the header and payload data is free for reuse by the application.

Figure 5 illustrates the storage of the headers and payload data in the TX pipeline 24. The Ethernet packet headers such as 50 together with the tags and pointers 51 are stored in the header FIFO buffer 33 in order of arrival whereas the corresponding payload data such as 53 are stored in the buffer space in the memory 32 allocated to one stream of packets. The whole of the payload data for a particular stream may be delivered as a single CLAN transfer direct from the application memory containing the data. Figure 4 illustrates the case where the payload data exceeds the maximum transfer unit size of Ethernet packets and thus has to be sliced into four payloads for transmission to the Ethernet as four separate Ethernet packets. The four headers for this transfer are illustrated in the buffer 33 in Figure 5 with the pointers of each packet indicating the location in the buffer 32 of the corresponding payload data for that packet.

Each CLAN PCI-link such as 6 in each of the computers of the cluster 5 includes a tripwire as disclosed in WO00/67131. The tripwire monitors incoming CLAN transfers and looks up each address in a CAM. If a match is found, the tripwire hardware sets a bit in a memory-mapped register and may also optionally raise an interrupt request. This technique is used to manage the flow of data into and out of the applications with a minimum number of interrupt requests. For example, it may not be necessary for every computer to respond instantly to the arrival every packet. In some case, it may be

sufficient for an application to poll the memory-mapped tripwire register to see if a buffer has been filled or emptied. Tripwires may also provide useful status information about which applications have work to do and may, for example, be used by a kernel to optimise dynamically a process scheduling algorithm.

It is thus possible to provide an arrangement which allows payload data from Ethernet packets to be put directly into user applications buffers so as to eliminate buffer-to-buffer copying. Similarly, a single transfer direct from memory of payload data for outgoing packets can be provided. Latency can therefore be substantially reduced and more CPU time can be devoted to user applications instead of being required for protocol stack processing. Use of the tripwire feature allows the number of interrupt requests to be substantially reduced with similar benefits. All computers in a CLAN cluster can use the network bridge with the same performance benefits.

Although a network bridge for bridging between a CLAN and a Gigabit Ethernet has been described, such a bridge may be used between other types of networks. For example, instead of a CLAN, the bridge may be used with SCI networks (or any network whose packets contain address information which addresses the final destination of the packet) and, instead of an Ethernet, the bridge may be used with an ATM network.

CLAIMS:

1 A network bridge for interfacing between a first network having a first protocol and a second network having a second protocol different from the first protocol, comprising: first means for receiving data packets from the first network, each data packet having a header and a data payload in accordance with the first protocol; and second means for adding to each of at least some of the received packets delivery data in accordance with the second protocol indicating the final destination address to which an application for processing the payload data of the packet requires delivery.

2 A network bridge as claimed in claim 1, in which the final destination address is the address in a randomly addressable device.

3 A network bridge as claimed in claim 2, in which the randomly addressable device is a memory.

4 A network bridge as claimed in any one of the preceding claims, in which the first network is an Ethernet.

5 A network bridge as claimed in any one of claims 1 to 3, in which the first network is an ATM network.

6 A network bridge as claimed in any one of the preceding claims, in which the second network is a memory mapped network.

7 A network bridge as claimed in any one of the preceding claim 6, in which the second network is a collapsed local area network.

8 A network bridge as claimed in any one of the preceding claims, in which the second means is arranged to remove at least part of the header from each of the at least some received packets.

9 A network bridge as claimed in claim 8, in which the at least part of the header comprises address data.

10 A network bridge as claimed in claim 8 or 9, in which the at least part of the header comprises all of the header

11 A network bridge as claimed in any one of the preceding claims, in which the second means is arranged to process at least part of the header of each of the at least some received packets so as to generate the delivery data.

12 A network bridge as claimed in claim 11, in which the second means contains a mapping from the at least part of the header of each of the at least some received packets to the delivery data, which mapping is programmable by at least one application in the second network.

13 A network bridge as claimed in claim 12, in which the second means comprises a content addressable memory arranged to address a random access memory.

14 A network bridge as claimed in any one of claims 11 to 13, in which the at least part of the header includes a sequence number of the payload data.

15 A network bridge as claimed in any one of the preceding claims, in which the second means is arranged, for the or each application, to direct the payload data of received packets to the correct position, relative to the positions of other payload data for the same application, in a payload re-assembly buffer of a computer on which the application runs, which re-assembly buffer is the final destination for the payload data.

16 A network bridge as claimed in claim 15, in which the second means is arranged to direct the headers of the received packets to a header storage buffer of a computer on which the application runs for storage in order of arrival of the received packets.

17 A network bridge as claimed in any one of the preceding claims, comprising: third means for receiving from the second network a plurality of packet headers in accordance with the first protocol; fourth means for receiving, independently of the headers, payload data from the second network; and fifth means for assembling the headers and the payload data into packets for transmission to the first network.

18 A network bridge as claimed in claim 17, in which the third means comprises a first-in/first-out buffer.

19 A network bridge as claimed in claim 17 or 18, in which each header received from the second network includes a pointer to a location in the fourth means for the corresponding payload data.

20 A network bridge as claimed in claim 19, in which the fifth means is arranged to remove the pointers before transmission to the first network.

21 A network bridge as claimed in any one of claims 17 to 20, comprising sixth means for calculating a checksum for each packet and for adding the checksum to the header.

22 A combination of a network bridge as claimed in any one of the preceding claims and a network switch for the second network having a first port connected to the network bridge.

23 A combination as claimed in claim 22, in which the network bridge is connected to the switch by an electrically conductive signal path.

24 A network comprising a combination as claimed in claim 22 or 23 and a plurality of computers, each of at least some of which is connected to a respective further port of the switch and has a final destination memory.

25 A network as claimed in claim 24, in which each of the at least some computers is connected to the switch by a respective optical fibre.

26 A network as claimed in claim 24 or 25 when dependent on any one of claims 17 to 21, in which at least one of the computers is arranged to run a first application which generates at least some of the packet headers in accordance with the first protocol.

27 A network as claimed in claim 26, in which the first application is arranged to reserve space in the fourth means for the payload data corresponding to the at least some packet headers in accordance with the first protocol.

28 A network as claimed in claim 26 or 27, in which the first application is arranged to supply the payload data by direct memory access.

29 A network as claimed in claim 28, in which the first application is arranged to request another application to supply the payload data by direct memory access.

30 A network as claimed in claim 28 or 29, in which the first application is arranged to request a device which is external to a central processing unit for running the first application to supply the payload data by direct memory access.

31 A network as claimed in claim 30, in which the device comprises a disk controller.

32 A network as claimed in any one of claims 24 to 31, in which at least one of the computers is arranged to run a second application which requires data from the first network.

33 A network as claimed in claim 32, in which the second application is arranged to provide a circular buffer for receiving payload data from the network bridge.

34 A network as claimed in claim 33, in which the application is arranged to provide a write pointer indicating a next position in the circular buffer for receiving the payload data and a read pointer indicating a position in the circular buffer where next data to be read are located.

35 A network as claimed in claim 34 when dependent on claim 15, in which the payload reassembly buffer is the circular buffer and the network bridge is arranged to update the write pointer when the circular buffer contains a contiguous block of data.

36 A network as claimed in claim 35, in which the second means is arranged to generate the delivery data from the write pointer and a packet sequence number.

37 A network as claimed in claim 35 or 36, at least one computer may comprise means for informing the application when the write pointer is updated.

38 A network as claimed in claim 37, in which the informing means comprises means for requesting an interrupt.

39 A network as claimed in claim 37, in which the informing means comprises means for setting a flag which is periodically polled by the application.

40 A network as claimed in any one of claims 34 to 39, in which the network bridge is arranged to compare the read and write pointers and to stop sending payload data to the circular buffer when the circular buffer is full.

41 A network bridge for interfacing between a first network having a first protocol and a second network having a second protocol different from the first protocol, comprising: first means for receiving from the second network a plurality of packet headers in accordance with the first protocol; second means for receiving, independently of the headers, payload data from the second network; and third means for assembling the headers and the payload data into packets for transmission to the first network.

42 A network bridge as claimed in claim 41, in which the first network is an Ethernet.

43 A network bridge as claimed in claim 41, in which the first network us an ATM network.

44 A network bridge as claimed in any one of claims 41 to 43, in which the second network is a memory mapped network.

45 A network bridge as claimed in claim 44, in which the second network is a collapsed local area network.

46 A network bridge as claimed in any one of claims 41 to 45, in which the first means comprises a first-in/first-out buffer.

47 A network bridge as claimed in any one of claims 41 to 46, in which each header received from the second network includes a pointer to a location in the second means for the corresponding payload data.

48 A network bridge as claimed in claim 47, in which the third means is arranged to remove the pointers before transmission to the first network.

49 A network bridge as claimed in any one of claims 41 to 48, comprising fourth means for calculating a checksum for each packet and for adding the checksum to the header.

50 A combination of a network bridge as claimed in any one of claims 41 to 49 and a network switch for the second network having a first port connected to the network bridge.

51 A combination as claimed in claim 50, in which the network bridge is connected to the switch by an electrically conductive signal path.

52 A network comprising a combination as claimed in claim 50 or 51, and a plurality of computers, each of at least some of which is connected to a respective further port of the switch.

53 A network as claimed in claim 52, in which each of the at least some computers is connected to the switch by a respective optical fibre.

54 A network as claimed in claim 52 or 53, in which at least one of the computers is arranged to run an application which generates at least some of the headers.

55 A network as claimed in claim 54, in which the application is arranged to reserve space in the fourth means for the payload data corresponding to the at least some headers.

56 A network as claimed in claim 54 or 55, in which the application is arranged to supply the payload data by direct memory access.

57 A network as claimed in claim 56, in which the application is arranged to request another application to supply the payload data by direct memory access.

58 A network as claimed in claim 56 or 57, in which the application is arranged to request a device which is external to a central processing unit for running the application to supply the payload data by direct memory access.

59 A network as claimed in claim 58, in which the device comprises a disk controller.

The Patent Office
INVESTOR IN PEOPLE

Application No: GB 0104788.5 **Examiner:** Mark Lewney
Claims searched: 1-40 **Date of search:** 16 November 2001

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.S): H4P (PF, PPA, PPEC)

Int Cl (Ed.7): H04L (12/46, 12/66, 29/06)

Other: Online databases: WPI, EPODOC, JAPIO.

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
X	GB2352146A	(FUJITSU LTD) - See abstract and fig 2.	1 at least
X	EP0594199A1	(DIGITAL EQUIPMENT) - See figs 4 & 5 and lines 24-26, col.11.	1, 4, 8, 9 at least
X	WO97/08838A2	(ERICSSON INC.) - See abstract.	1 at least.
X	US6108345	(3COM CORPORATION) - See abstract, lines 52-64 in column 2 and fig. 8.	1, 7-10 at least.
X	US5651002	(3COM CORPORATION) See abstract, col.2 lines 34-38, col. 5 lines 39-44, col. 6 lines 34-44.	1-4, 7, 15, 16 at least.

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.